```
// adjust_task_sla.txt
// Copyright (c) 2003. Sychron, Inc. All Rights Reserved.
 1: int swm_jobs_adjust_task_sla(swm_task_t *task) {
 2:   swm_job_sla_t desired_delta, actual_delta, zero_delta;
 3:   swm_jd_t *jd;
 4:   int index;
 5:   /*-- 1. Calculate desired change in the task actual SLA --*/
 6:   desired_delta.cpu = task->desired_sla.cpu - task->min_cpu;
 7:   desired_delta.comm = task->desired_sla.comm - task->min_comm;
 8:   desired_delta.in_bandwidth =
 9:     task->desired_sla.in_bandwidth - task->in_bandwidth;
10:   desired_delta.out_bandwidth =
11:     task->desired_sla.out_bandwidth - task->out_bandwidth;
12:   if (desired_delta.cpu == 0 && desired_delta.comm == 0 &&
13:       desired_delta.in_bandwidth == 0 &&
   desired_delta.out_bandwidth == 0) {
14:     slog_msg(SLOG_DEBUG, "debug: no SLA change required for
   job %llu",
15:              task->job_id);
16:     return 0;
17:   }
18:
19:   /*-- 2. Debugging info --*/
20:
21:   slog_msg(SLOG_DEBUG, "debug: adjusting SLA of job %llu from
   (%u CPU, "
22:            "%u comms, %u in, %u out) to (%u CPU, %u comms,
   %u in, %u out)",
23:            task->job_id, (unsigned) task->min_cpu, (unsigned)
   task->min_comm,
24:            (unsigned) task->in_bandwidth, (unsigned)
   task->out_bandwidth,
25:            (unsigned) task->desired_sla.cpu, (unsigned)
   task->desired_sla.comm,
26:            (unsigned) task->desired_sla.in_bandwidth,
27:            (unsigned) task->desired_sla.out_bandwidth);
28:
29:   /*-- 3. Mark task as non-rogue --*/
30:   task->flags &= ~SWM_JOB_ROGUE;
31:
32:   /*-- 4. Change resource bookings --*/
33:   swm_grid_change_resource_bookings(&desired_delta, &actual_delta);
34:   slog_msg(SLOG_DEBUG, "debug: actual delta CPU %d, comm %d,
   in %d, out %d",
35:            (int) actual_delta.cpu, (int) actual_delta.comm,
36:            (int) actual_delta.in_bandwidth,
37:            (int) actual_delta.out_bandwidth);
38:
39:   /*-- 5. Raise alarm if desired SLA cannot be realised --*/
40:   if (memcmp(&desired_delta, &actual_delta, sizeof(desired_delta))
```

```
  != 0)
41:     lswm_jobs_insufficient_resources(task->job_id);
42:
43:   /*-- 6. Exit if no change at all was possible --*/
44:   memset(&zero_delta, 0, sizeof(zero_delta));
45:   if (memcmp(&zero_delta, &actual_delta, sizeof(zero_delta)) == 0)
46:     return 0;
47:
48:   /*-- 7. Update the job descriptor --*/
49:
50:   /*-- 7.1. Get job descriptor, reverting changes if an error
  occurs --*/
51:   if ((jd = swm_jobs_get_descriptor(task->job_id, SDB_UPDATE_LOCK))
  == NULL) {
52:     slog_msg(SLOG_WARNING, "Cannot get descriptor for job %llu in "
53:             "swm_jobs_adjust_task_sla", task->job_id);
54:     desired_delta.cpu = -actual_delta.cpu;
55:     desired_delta.comm = -actual_delta.comm;
56:     desired_delta.in_bandwidth = -actual_delta.in_bandwidth;
57:     desired_delta.out_bandwidth = -actual_delta.out_bandwidth;
58:     swm_grid_change_resource_bookings(&desired_delta,
  &actual_delta);
59:     return 0;
60:   }
61:
62:   /*-- 7.2. Modify SLAs in the job descriptor; mark job as
  non-rogue --*/
63:   if (!(jd->flags & SWM_JOB_ADJUSTED))
64:     jd->desired_instance_sla = task->desired_sla;
65:   jd->actual_sla.cpu += actual_delta.cpu;
66:   jd->actual_sla.comm += actual_delta.comm;
67:   jd->actual_sla.in_bandwidth += actual_delta.in_bandwidth;
68:   jd->actual_sla.out_bandwidth += actual_delta.out_bandwidth;
69:   index = task->logic_node_id;
70:   jd->actual_instance_slas[index].cpu += actual_delta.cpu;
71:   jd->actual_instance_slas[index].comm += actual_delta.comm;
72:   jd->actual_instance_slas[index].in_bandwidth +=
  actual_delta.in_bandwidth;
73:   jd->actual_instance_slas[index].out_bandwidth +=
  actual_delta.out_bandwidth;
74:   jd->flags &= ~SWM_JOB_ROGUE;
75:   if (swm_jobs_set_descriptor(jd, SDB_UNLOCK) < 0) {
76:     slog_msg(SLOG_WARNING, "Cannot set descriptor for job %llu in "
77:             "swm_jobs_adjust_task_sla", task->job_id);
78:     desired_delta.cpu = -actual_delta.cpu;
79:     desired_delta.comm = -actual_delta.comm;
80:     desired_delta.in_bandwidth = -actual_delta.in_bandwidth;
81:     desired_delta.out_bandwidth = -actual_delta.out_bandwidth;
82:     swm_grid_change_resource_bookings(&desired_delta,
  &actual_delta);
```

```
83:     swm_jobs_free_descriptor(jd);
84:     return 0;
85:   }
86:
87:   /*-- 7.3. Free job descriptor --*/
88:   swm_jobs_free_descriptor(jd);
89:
90:   /*-- 8. Update task actual SLA --*/
91:   task->min_cpu += actual_delta.cpu;
92:   task->min_comm += actual_delta.comm;
93:   task->in_bandwidth += actual_delta.in_bandwidth;
94:   task->out_bandwidth += actual_delta.out_bandwidth;
95:
96:   /*-- 9. Set CPU and bandwidth SLAs --*/
97:   if (actual_delta.cpu != 0 && task->pids.npids > 0 &&
98:       sds_set_contract(swm_sei_fd, (pid_t)
    task->pids.pids[0].data.id,
99:                        (double)task->min_cpu /
    (double)swm_resources.cpu) < 0)
100:     slog_msg(SLOG_WARNING, "SDS continual contract failed for
    job %llu proc %d"
101:              " (errno %d: %s)", task->job_id, (int)
    task->pids.pids[0].data.id,
102:              errno, strerror(errno));
103:   if (actual_delta.comm != 0 &&
104:       lswm_set_bwman_sla_pipe(task,
105:                               SYC_BWMAN_PIPE_ID_INTERNAL,
106:                               0, task->min_comm) < 0)
107:     slog_msg(SLOG_WARNING, "Cannot set bwman SLA pipe INT for
    job %llu (%s)",
108:              task->job_id, strerror(errno));
109:   if ((actual_delta.in_bandwidth != 0 ||
    actual_delta.out_bandwidth != 0) &&
110:       lswm_set_bwman_sla_pipe(task,
111:                               SYC_BWMAN_PIPE_ID_EXTERNAL,
112:                               0, task->out_bandwidth) < 0)
113:       slog_msg(SLOG_WARNING, "Cannot set bwman SLA pipe EXT for
    job %llu (%s)",
114:                task->job_id, strerror(errno));
115:
116:   /*-- 10. Return TRUE --*/
117:   return 1;
118: }
// info_exch_alarm.txt
// Copyright (c) 2003. Sychron, Inc. All Rights Reserved.
 1: void sychron_info_exch_alarm(void *alarm_data) {
 2:   sos_clock_t  time_now;
 3:   int          i, j, rc;
 4:   syc_uint32_t backoff_rand;
 5:
```

```
 6:  if (!syc_info_exch)
 7:    return;
 8:
 9:
10:  if (syc_info_exch->this_nodeid == SYCHRON_MAX_CLUSTER_NODES) {
11:    syc_info_exch->this_nodeid = sychron_server_myself();
12:    if (syc_info_exch->this_nodeid == SYCHRON_MAX_CLUSTER_NODES) {
13:      return;
14:    }
15:    slog_msg(SLOG_INFO,"sychron_info_exch(node %d
  initialising  random seed)",
16:       syc_info_exch->this_nodeid);
17:    syc_irand_seed(2*syc_info_exch->this_nodeid+1,
18:       &syc_info_exch->random_state);
19:  }
20:  sychron_server_active(syc_info_exch->active_nodes);
21:  syc_info_exch->master_nodeid=
  syc_nodeset_first(syc_info_exch->active_nodes);
22:  sychron_info_exch_parameters();
23:
24:  backoff_rand = syc_irand(&syc_info_exch->random_state);
25:  time_now     = sos_clocknow();
26:  for(i=0;i<=SYCHRON_INFO_EXCH_CODE_MAX; i++) {
27:    syc_info_exch_reg_t *reg = syc_info_exch->registrations[i];
28:    if (reg) {
29:      if (reg->next_trigger <= time_now) {
30:  syc_info_exch_reg_data_t *info;
31:
32:  info = &reg->most_recent[syc_info_exch->this_nodeid];
33:  reg->next_trigger += reg->gap;
34:
35:       sychron_info_exch_reg_attempt_critical_section(reg,{
36:    rc = reg->send(i,info->data,&info->nbytes);
37:    if (rc)
38:      reg->stats.send_callback_nochange++;
39:    else {
40:      reg->retransmit_range = 1;
41:      reg->retransmit_count = 0;
42:      reg->stats.send_callback_change++;
43:      syc_nodeset_set(reg->most_recent_recvd,
  syc_info_exch->this_nodeid);
44:      info->timestamp      = time_now;
45:      info->ttl            = syc_info_exch->max_ttl;
46:      SYC_ASSERT(info->nbytes <= reg->max_nbytes);
47:    }
48:
49:         if ((reg->flags &
  SYCHRON_INFO_EXCH_FLAGS_CHECKPOINT_DATA) &&
50:       (syc_info_exch->master_nodeid ==
  syc_info_exch->this_nodeid)) {
```

```
51:      syc_nodeset_iterate(syc_info_exch->active_nodes,j,{
52:        syc_info_exch_reg_data_t *info_j   = &reg->most_recent[j];
53:        syc_info_exch_reg_data_t *frozen_j = &reg->frozen[j];
54:
55:        if (info_j->timestamp > frozen_j->timestamp) {
56:     /* make sure this node sees the frozen entries */
57:     syc_nodeset_set(reg->frozen_recvd,j);
58:     frozen_j->timestamp = info_j->timestamp;
59:     frozen_j->ttl       = info_j->ttl;
60:     frozen_j->nbytes    = info_j->nbytes;
61:     if (info_j->nbytes)
62:       syc_memcpy(frozen_j->data,info_j->data,info_j->nbytes);
63:        }
64:      });
65:    }
66:  });
67:
68:        if (!reg->uploaded)
69:     sos_task_schedule_asap(&reg->recv_all_task_reset_recv);
70:   else
71:     sychron_info_exch_recv_all_task_reset_recv(reg);
72:
73:        if (!info->ttl) {
74:     if (reg->retransmit_count)
75:       reg->retransmit_count--;
76:     else {
77:       /* Updating ttl forces retransmit */
78:       info->ttl = syc_info_exch->max_ttl;
79:       if (reg->retransmit_range < SYC_INFO_EXCH_MAX_RETRANSMIT_PERIOD)
80:         reg->retransmit_range
81:     = syc_min(2*reg->retransmit_range,
82:         SYC_INFO_EXCH_MAX_RETRANSMIT_PERIOD);
83:       if (!reg->retransmit_range)
84:         reg->retransmit_range = SYC_INFO_EXCH_MAX_RETRANSMIT_PERIOD;
85:       reg->retransmit_count = backoff_rand %
86:             ((syc_uint32_t)reg->retransmit_range);
87:     }
88:   } /* zero ttl */
89:      } /* registration update period */
90:    } /* active registration */
91:  } /* forall registrations */
92:
93:   sos_task_schedule_asap(&syc_info_exch->send_task);
94: }
```

```
// info_exch_prepare_send.txt
```



```
 1: STATIC int sychron_info_exch_prepare_send(syc_uint16_t master_ttl,
 2:             syc_nodeid_t ignore_node) {
 3:   syc_info_exch_pkt_reg_header_t *data_header;
 4:   syc_info_exch_reg_t            *reg;
```

```
 5:  syc_info_exch_reg_data_t       *info;
 6:  int                            rc, i, j, elems_tot, nbytes,
  misaligned,
 7:                                 reg_code, freeze, no_frozen;
 8:  char                           *payload;
 9:  syc_nodeset_t                  send_set;
10:  syc_nodeset_t                  transfer_set;
11:  syc_info_exch_pkt_header_t     *packet;
12:  static syc_uint32_t            reg_fairness = 0;
13:
14:  packet = sqi_qcell_alloc(&syc_info_exch->packet_allocator);
15:  if (!packet)
16:    return -ENOMEM;
17:  no_frozen = 0;
18:  elems_tot = 0;
19:  nbytes    = sizeof(syc_info_exch_pkt_header_t);
20:  data_header
21:    = (syc_info_exch_pkt_reg_header_t*) (packet+1);
22:  SYC_ASSERT(!(((syc_uintptr_t) data_header) %
  SYC_BASIC_TYPE_ALIGNMENT));
23:
24:  reg_fairness = (reg_fairness + 1) %
  (SYCHRON_INFO_EXCH_CODE_MAX+1);
25:  for(i=0;i<=SYCHRON_INFO_EXCH_CODE_MAX; i++) {
26:    reg_code = (reg_fairness + i) %
  (SYCHRON_INFO_EXCH_CODE_MAX+1);
27:    reg = syc_info_exch->registrations[reg_code];
28:    if (reg) {
29:
30:      syc_nodeset_zeros(send_set);
31:      syc_nodeset_iterate(syc_info_exch->active_nodes,j,{
32:  freeze = master_ttl && (reg->flags &
33:              SYCHRON_INFO_EXCH_FLAGS_CHECKPOINT_DATA);
34:  info   = (freeze)?&reg->frozen[j]:&reg->most_recent[j];
35:  if (info->timestamp && info->ttl) {
36:    no_frozen += freeze;
37:    syc_nodeset_set(send_set,j);
38:  }
39:      });
40:
41:      /* no updates... onto the next registration */
42:      if (syc_nodeset_isempty(send_set))
43:  continue;
44:
45:      {
46:  int thisnode_set, max_elems;
47:
48:  thisnode_set =
  syc_nodeset_isset(send_set,syc_info_exch->this_nodeid);
49:
```

```
50:   if (thisnode_set)
51:     syc_nodeset_clr(send_set,syc_info_exch->this_nodeid);
52:
53:   if (master_ttl &&
54:     (reg->flags & SYCHRON_INFO_EXCH_FLAGS_CHECKPOINT_DATA))
55:     max_elems = (SYC_INFO_EXCH_MAX_MTU - nbytes) /
56:                 (reg->max_nbytes +
57:           sizeof(syc_info_exch_pkt_reg_header_t) +
58:           SYC_BASIC_TYPE_ALIGNMENT);
59:   else
60:     max_elems = syc_info_exch->max_elems_send;
61:   syc_nodeset_random_subset(&syc_info_exch->random_state,
62:           send_set,
63:           transfer_set,
64:           (max_elems > 1)?max_elems-1:1);
65:   if (thisnode_set)
66:     syc_nodeset_set(transfer_set,syc_info_exch->this_nodeid);
67:       }
68:
69:       /* Make sure data does not change under our feet */
70:       sychron_info_exch_reg_attempt_critical_section(reg,{
71:   syc_nodeset_iterate(transfer_set,j,{
72:     info = (master_ttl && (reg->flags &
73:           SYCHRON_INFO_EXCH_FLAGS_CHECKPOINT_DATA))?
74:             &reg->frozen[j]:&reg->most_recent[j];
75:     if ((elems_tot >= 255)              ||
    /* syc_uint8_t in payload*/
76:         (nbytes +
77:           sizeof(syc_info_exch_pkt_reg_header_t) +
78:           info->nbytes +
79:           SYC_BASIC_TYPE_ALIGNMENT) > SYC_INFO_EXCH_MAX_MTU)
80:       syc_info_exch->stats.send_truncated++;
81:
82:     else if (info->timestamp && info->ttl) {
83:       data_header->ttl       = info->ttl--;
84:       data_header->timestamp = info->timestamp;
85:       data_header->nodeid    = j;
86:       data_header->reg_code  = reg_code;
87:       data_header->nbytes    = info->nbytes;
88:       payload = (char*) (data_header+1);
89:       if (info->nbytes)
90:         syc_memcpy(payload,info->data,info->nbytes);
91:       misaligned =
   (info->nbytes+sizeof(syc_info_exch_pkt_reg_header_t))%
92:         SYC_BASIC_TYPE_ALIGNMENT;
93:       data_header->padding =
94:         !misaligned?0:SYC_BASIC_TYPE_ALIGNMENT-misaligned;
95:       nbytes += sizeof(syc_info_exch_pkt_reg_header_t) +
96:                 data_header->nbytes + data_header->padding;
97:       data_header
```

```
98:       = (syc_info_exch_pkt_reg_header_t*) (payload +
99:             data_header->nbytes +
100:              data_header->padding);
101:     elems_tot++;
102:     reg->stats.send_updates[j]++;
103:   }
104: });
105:    });
106:   } /* active registration */
107: } /* for each registration */
108:
109: packet->elems      = elems_tot;
110: packet->nbytes     = nbytes;
111: packet->master_ttl = no_frozen?master_ttl:0;
112: packet->via        = syc_info_exch->this_nodeid;
113:
114: if (!elems_tot)
115:   rc = -ENODATA;
116:
117: else {
118:   syc_nodeset_copy(send_set,syc_info_exch->active_nodes);
119:   syc_nodeset_clr(send_set,syc_info_exch->this_nodeid);
120:   syc_nodeset_clr(send_set,ignore_node);
121:   if (master_ttl)
122:     syc_nodeset_clr(send_set,syc_info_exch->master_nodeid);
123:   rc = sychron_info_exch_send(packet,
124:       send_set,
125:       (master_ttl &&
126:        (master_ttl < syc_info_exch->max_ttl))?
127:       syc_info_exch->rest_mates:
128:       syc_info_exch->first_mates);
129: }
130: sqi_qcell_free(&syc_info_exch->packet_allocator,packet);
131: return rc;
132: }
// info_exch_send.txt
// Copyright (c) 2003. Sychron, Inc. All Rights Reserved.
 1: STATIC int sychron_info_exch_send(syc_info_exch_pkt_header_t *pkt,
 2:         syc_nodeset_t             send_set,
 3:         int                       no_mates) {
 4:  syc_nodeset_t   mates;
 5:  int            i, j, rc, nbytes;
 6:  sos_netif_pkt_t raw_pkt;
 7:
 8:  syc_nodeset_random_subset(&syc_info_exch->random_state,
 9:         send_set,
10:         mates,
11:         no_mates);
12:  nbytes = pkt->nbytes;
13:  j = 0;
```

```
14:   syc_nodeset_iterate(mates,i,{
15:     raw_pkt.syc_proto = SOS_NETIF_INFOX;
16:     raw_pkt.proto.sychron.nodeid = i;
17:     raw_pkt.total_len = nbytes;
18:     raw_pkt.num_vec   = 1;
19:     raw_pkt.vec_in_place.iov_base = (void*)pkt;
20:     raw_pkt.vec_in_place.iov_len  = nbytes;
21:     raw_pkt.vec                   = &raw_pkt.vec_in_place;
22:     rc = sos_netif_bottom_tx(&raw_pkt);
23:     if (rc)
24:       syc_info_exch->stats.send_pkts_fail[i]++;
25:
26:     else {
27:       j++;
28:       syc_info_exch->stats.send_pkts[i]++;
29:       syc_info_exch->stats.send_bytes[i] += nbytes;
30:       if (pkt->master_ttl) {
31:   syc_info_exch->stats.send_master_pkts[i]++;
32:   syc_info_exch->stats.send_master_bytes[i] += nbytes;
33:       }
34:     }
35:   });
36:   return (j?0:-ENODATA);
37: }
// move_detected_process.txt
// Copyright (c) 2003. Sychron, Inc. All Rights Reserved.
 1: void swm_jobs_move_detected_process(swm_job_id_t job_id,
 2:                                     swm_process_obj_t process,
 3:                                     int include_child_pids,
 4:                                     swm_job_sla_t *sla,
 5:                                     char *job_name,
 6:                                     int detection_level) {
 7:   swm_task_t *task;
 8:
 9:   /*-- 1. Place the process in the appropriate task --*/
10:   if ((task = lswm_jobs_place_process(job_id, job_name, process,
11:                                       detection_level)) == NULL)
12:     return;
13:
14:   /*-- 2. Update task flags --*/
15:   if (include_child_pids)
16:     task->flags |= SWM_JOB_MANAGED;
17:   else
18:     task->flags &= ~SWM_JOB_MANAGED;
19:
20:   /*-- 3. Give the task the appropriate SLA --*/
21:   if (sla && !(task->flags & SWM_JOB_ADJUSTED) &&
22:       memcmp(&task->desired_sla, sla, sizeof(*sla)) != 0) {
23:     task->desired_sla = *sla;
24:     if (swm_jobs_adjust_task_sla(task) == 0)
```

```
25:       lswm_update_desired_instance_sla(task);
26:   }
27: }
// pipe_tx_info_exch_recv_all.txt
// Copyright (c) 2003. Sychron, Inc. All Rights Reserved.
 1: void syc_bwman_shared_pipe_tx_info_exch_recv_all
   (syc_uint8_t   reg_code,
 2:          void       **buffer_arr,
 3:          syc_uint16_t *nbytes_arr,
 4:            syc_nodeset_t active) {
 5:   syc_int32_t total_bytes_booked;
 6:   syc_int32_t total_bytes_unbooked;
 7:   syc_int32_t total_bytes_queued;
 8:   syc_bwman_pipe_id_t pipe_id;
 9:   syc_bwman_shared_pipe_t *pipe;
10:   syc_nodeid_t nodeid;
11:   syc_bwman_shared_pipe_config_t *pipe_config ;
12:   syc_bwman_shared_pipe_info_exch_t *my_recv_state;
13:   syc_uint32_t num_active_nodes ;
14:
15:   pipe_id = reg_code - SYCHRON_INFO_EXCH_CODE_STROBE_PIPE_MIN;
16:   if (pipe_id < 0 ||
17:       reg_code > SYCHRON_INFO_EXCH_CODE_STROBE_PIPE_MAX ||
18:       pipe_id >= SYC_BWMAN_MAX_PIPES) {
19:
20:     static sos_clock_t  time_next_error = 0;
21:     static syc_uint32_t error_count = 0;
22:
23:     error_count++;
24:     if (sos_clocknow() > time_next_error) {
25:       slog_msg(SLOG_CRIT,
26:          "syc_bwman_shared_pipe_recv_all(unknown pipe reg %d)
   errors=%d",
27:          reg_code,error_count);
28:     }
29:     return;
30:   }
31:   pipe = &pipe_table[pipe_id];
32:   my_recv_state = &pipe->recv_state[pipe_nodeid];
33:   pipe_config = &pipe_config_table[pipe_id];
34:
35:   syc_nodeset_iterate(active,nodeid,{
36:     if (nbytes_arr[nodeid] !=
   sizeof(syc_bwman_shared_pipe_info_exch_t))
37:       slog_msg(SLOG_CRIT,"syc_bwman_shared_pipe_recv_all(INVARIANT
   from %d)",
38:          nodeid);
39:     else
40:       syc_memcpy(&pipe->recv_state[nodeid],buffer_arr[nodeid],
   nbytes_arr[nodeid]);
```

```
41:  });
42:
43:  total_bytes_booked           = 0;
44:  total_bytes_unbooked         = 0;
45:  syc_nodeset_iterate(active_nodes,nodeid,{
46:    syc_bwman_shared_pipe_info_exch_t *recv_state =
47:      &pipe->recv_state[nodeid];
48:
49:    total_bytes_booked   += recv_state->bytes_booked;
50:    total_bytes_unbooked += recv_state->bytes_unbooked;
51:  });
52:
53:  total_bytes_queued = total_bytes_unbooked + total_bytes_booked;
54:
55:  syc_bwman_desc_add_booked_credit(pipe_config);
56:
57:
58:  num_active_nodes = syc_nodeset_count(active_nodes);
59:  if (num_active_nodes==0) num_active_nodes = 1;
60:
61:  pipe_config->shared.nonqueued_bytes =
62:    (pipe_config->total_bytes > total_bytes_queued) ?
63:    pipe_config->total_bytes - total_bytes_queued : 0 ;
64:
65:  {
66:    syc_uint32_t available_this_time =
67:      pipe_config->shared.nonqueued_bytes / num_active_nodes ;
68:
69:    syc_uint32_t unused_last_time =
70:      sci_atomic_swap32((syc_uint32_t*)&pipe_config->
  my.nonqueued_counter,
71:      available_this_time) ;
72:
73:    if (pipe_config->my.overdrawn_counter > 0) {
74:      (void)sci_atomic_add32((syc_int32_t*)&pipe_config->
  my.overdrawn_counter,
75:           - unused_last_time );
76:      if (pipe_config->my.overdrawn_counter < 0)
77:  pipe_config->my.overdrawn_counter = 0;
78:    }
79:  }
80:
81:  if (pipe_config->shared.nonqueued_bytes > 0) {
82:    pipe_config->unbooked_share = SYC_BWMAN_PIPE_SHARE_MAX;
83:    syc_bwman_desc_add_unbooked_credit(pipe_config);
84:
85:    pipe_config->immediate_credits =
86:      x_times_y_div_z((pipe_config->shared.nonqueued_bytes),
87:          pipe_config->immediate_share,
88:          SYC_BWMAN_PIPE_SHARE_MAX)
```

```
89:      / num_active_nodes ;
90:    syc_bwman_desc_add_immediate_credit(pipe_config);
91:  } else {
92:    pipe_config->immediate_credits = 0 ;
93:
94:    // Testing that total_bytes_unbooked >0 is theoretically
95:    // superfluous, but the code is complex and makes it hard to
96:    // verify that div by 0 is not possible.
97:
98:  if ((total_bytes_booked < pipe_config->total_bytes) &&
99:  (total_bytes_unbooked > 0)) {
100:      pipe_config->unbooked_share =
101:  x_times_y_div_z( pipe_config->total_bytes - total_bytes_booked,
102:       SYC_BWMAN_PIPE_SHARE_MAX,
103:       total_bytes_unbooked);
104:      syc_bwman_desc_add_unbooked_credit(pipe_config);
105:    } else
106:      pipe_config->unbooked_share = 0;
107:  }
108: #ifdef SYC_BWMAN_SHARED_PIPE_STATS
109:  pipe->stats.total_bytes        = pipe_config->total_bytes ;
110:  pipe->stats.sent_booked_bytes   = total_bytes_booked ;
111:  pipe->stats.sent_unbooked_bytes =
112:    x_times_y_div_z(total_bytes_unbooked,
113:       pipe_config->unbooked_share,
114:       SYC_BWMAN_PIPE_SHARE_MAX);
115:  pipe->stats.immediate_bytes     = pipe_config->
   immediate_credits ;
116: #endif
117:
118:  syc_bwman_pipe_tx_pkts(pipe_config);
119: }
// pipe_tx_info_exch_send.txt
// Copyright (c) 2003. Sychron, Inc. All Rights Reserved.
 1: int syc_bwman_shared_pipe_tx_info_exch_send(syc_uint8_t reg_code,
 2:              void        *send_buffer,
 3:              syc_uint16_t *send_nbytes) {
 4:  syc_bwman_pipe_id_t      pipe_id;
 5:  syc_bwman_shared_pipe_t *pipe;
 6:  syc_bwman_shared_pipe_config_t *pipe_config ;
 7:
 8:  pipe_id = reg_code - SYCHRON_INFO_EXCH_CODE_STROBE_PIPE_MIN;
 9:  if (pipe_id < 0 ||
10:      reg_code > SYCHRON_INFO_EXCH_CODE_STROBE_PIPE_MAX ||
11:      pipe_id >= SYC_BWMAN_MAX_PIPES) {
12:    static sos_clock_t  time_next_error = 0;
13:    static syc_uint32_t error_count = 0;
14:
15:    error_count++;
16:    if (sos_clocknow() > time_next_error) {
```

```
17:       slog_msg(SLOG_CRIT,
18:           "syc_bwman_shared_pipe_tx_info_exch_send(unknown pipe reg
  %d) errors=%d",
19:           reg_code, error_count);
20:       time_next_error = sos_clocknow() + sos_clock_from_secs(5);
21:     }
22:     return -EINVAL;
23:   }
24:   pipe_config = &pipe_config_table[pipe_id];
25:   sychron_server_active(active_nodes);
26:
27:   syc_bwman_desc_booking_summary(pipe_config);
28:
29: \*
30: Update the information which will be shared with the other nodes.
31: *\
32:   pipe = &pipe_table[pipe_id];
33:   pipe->my_send_state.bytes_unbooked = pipe_config->total_unbooked;
34:   pipe->my_send_state.bytes_booked   = pipe_config->total_booked;
35:   if (pipe_config->total_booked > pipe_config->total_bytes) {
36:     static sos_clock_t  time_next_error = 0;
37:     static syc_uint32_t error_count = 0;
38:
39:     error_count++;
40:     if (sos_clocknow() > time_next_error) {
41:       slog_msg(SLOG_WARNING,
  "syc_bwman_shared_pipe_tx_info_exch_send(INVARIANT: "
42:           "booked=%d > total_bytes=%d errors=%d)",
43:           pipe_config->total_booked,
44:           pipe_config->total_bytes,
45:           error_count);
46:       time_next_error = sos_clocknow() + sos_clock_from_secs(5);
47:     }
48:   }
49:   if (!syc_bwman_shared_pipe_delta(&pipe->my_send_state,
50:             &pipe->recv_state[pipe_nodeid])) {
51:     pipe->stats.tx_schedules_send_nochange++;
52:     return 1;
53:   } else {
54:     pipe->stats.tx_schedules_send_change++;
55:     syc_memcpy(send_buffer,
56:           &pipe->my_send_state,
57:           sizeof(syc_bwman_shared_pipe_info_exch_t));
58:     *send_nbytes = sizeof(syc_bwman_shared_pipe_info_exch_t);
59:     return 0;
60:   }
61: }
```